

Performance Portability with OpenMP: Experiences with 4.5, Looking Toward 5.0

Tom Scogland



LLNL-PRES-689578

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC



OpenMP 4.0/4.5 Feature Overview

- **Significantly improved support for devices (accelerators, etc.)** OpenMP now provides mechanisms for unstructured data mapping, asynchronous execution and also runtime routines for device memory management. These routines allow for explicit allocation, copying and freeing of memory between devices.
- **Support for doacross loops.** A natural mechanism to parallelize loops with well-structured dependences is provided.
- **New taskloop construct.** Support to divide loops into tasks, avoiding the requirement that all threads execute the loop.
- **Reductions for C/C++ arrays.** This often requested feature is now available by building on support for array sections.

OpenMP 4.0/4.5 Feature Overview

Cont.

- **Task and lock hint mechanisms.** Hint mechanisms can provide guidance on the relative priority of tasks and on preferred synchronization implementations.
- **Thread affinity support.** It is now possible to use runtime functions to determine the effect of thread affinity clauses.
- **Improved support for Fortran 2003.** Users can now parallelize many Fortran 2003 programs.
- **SIMD extensions.** These extensions include the ability to specify exact SIMD width and additional data-sharing attributes.

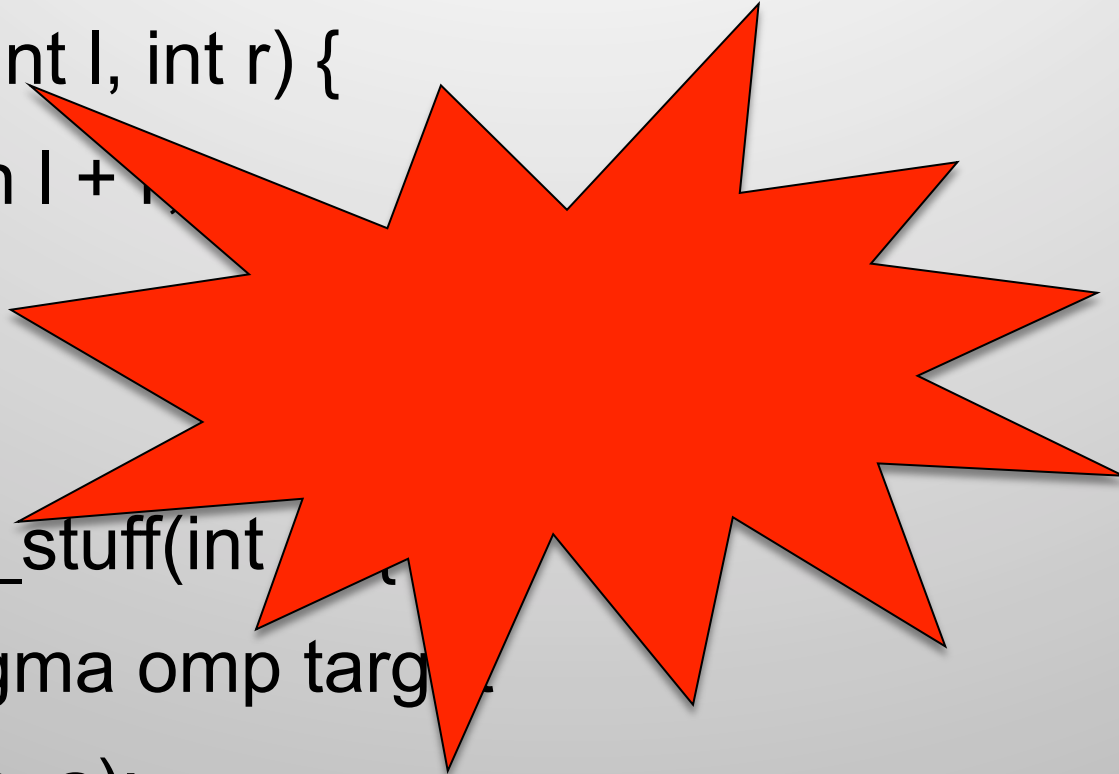
**Before performance portability, we
need expressibility, and functional
portability**

Biggest “surprises” for application developers in 4.X

Simple Example #1

```
int add(int l, int r) {  
    return l + r;  
}
```

```
void do_stuff(int a) {  
    #pragma omp target  
    add(a, a);  
}
```



In 4.0-4.5:

All Code Offloaded *Must* be Marked

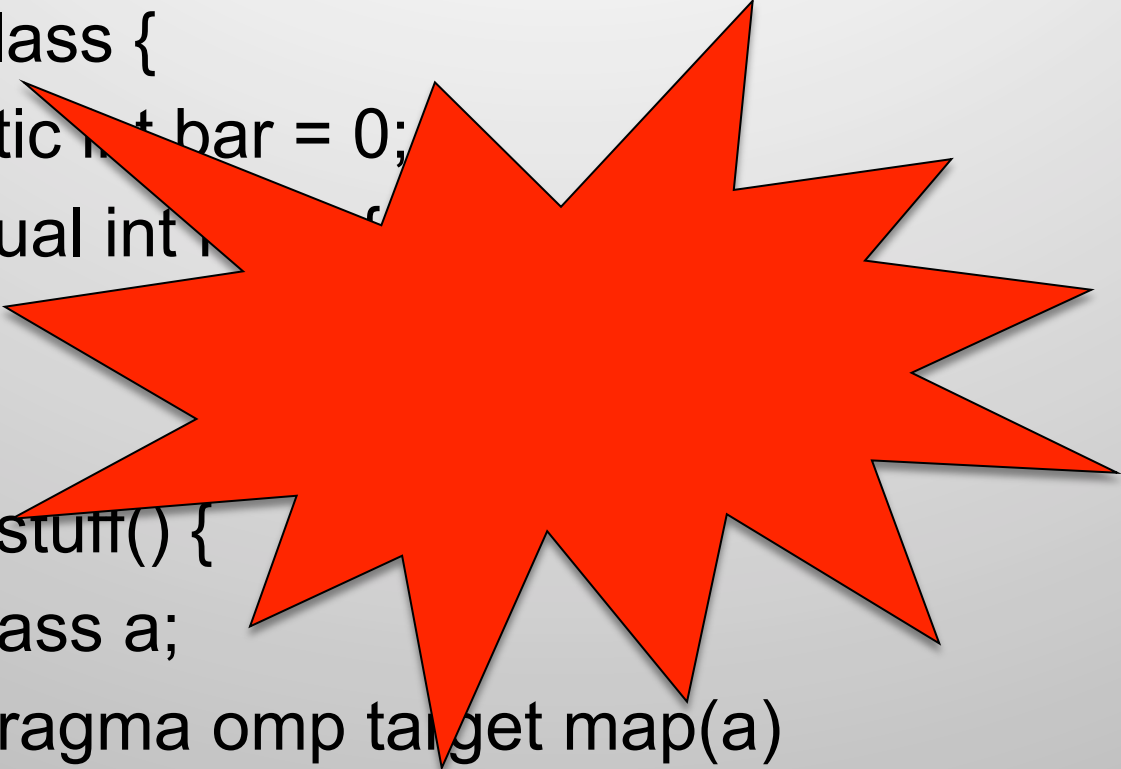
- All functions called in target regions ***must*** be marked `declare target`
- Classic routines are not available on the device:
 - `assert`
 - `exit`
 - `MPI_*`
 - `etc.`
- All user code must be marked, libraries must be updated

Future: Automatic Propagation of `declare target`

- Observation: Many functions that one calls in device code may be available in the same translation unit, anything with:
 - `static`
 - `constexpr`
 - `template`
- Extension: Explicitly allow such functions to not be marked, let the compiler do the work

Simple Example #2

```
class Vclass {  
    static int bar = 0;  
    virtual int f()  
};  
  
void do_stuff() {  
    Vclass a;  
    #pragma omp target map(a)  
    a.foo();  
}
```



In 4.0-4.5: Some C++ Features are not Enabled

- Virtual methods on objects instantiated on the host likely will not work in device code
- STL classes are not marked declare target
 - They're not available now and may not be later
- Dynamic allocation may be unavailable, or severely limited

C++ and Fortran Class Support is Under-Specified

- Mapping of:
 - Member variables (especially private)
 - “this” inside a method
 - Classes with non-trivial constructors
- Objects of classes with static or virtual members *cannot* be mapped
 - The static members themselves can...

Simple Example #3

```
struct a { int *a, intptr b} a1 = { (int[50]){0}, 5};  
#pragma omp target  
a1.b = a1.a  
printf("%p\n", a1.b);
```

Result: NULL...



In 4.0-4.5: Mapping Data can be Counter-Intuitive and Tedious

- There are no built-in facilities for deep-copy (yet)

```
struct a { int *a, int b} a1 = { (int[50]){0}, 5};  
#pragma omp parallel for ...
```

Becomes:

```
struct a { int *a, int b} a1 = { (int[50]){0}, 5};  
#pragma omp target teams distribute parallel for  
map(a1, a1.a[50]) ...
```

Future:

Mapping and Data Description

- Deep copy will be coming
- Annotations, or user-defined mechanisms, will allow easier management of complex data
- Mapping, by bitwise copy, of objects with these are likely to be supported as well:
 - static members
 - virtual members
 - parameters

What does this mean for porting applications?

Applications Will Need new Programming Patterns

- Many codes will need significant refactoring to take advantage of GPUs and others with OpenMP 4.5
- Upside: Most of these optimizations will also make the CPU versions more efficient

Third-Party Library Use Will be Tricky

- Functions, classes and structures used in target regions must be written to OpenMP or the underlying model to be available
 - Exception: Template or header-based libraries in C++
- Major libraries will need reworks to play nice:
 - Hypre
 - ScaLAPACK
 - HDF
 - etc.

**Once a code is ported, is it
performance portable?**

Performance Portability is a Concern

- Using hints to optimize a given platform may hurt performance on another platform
- At present there is no facility for targeting optimizations to a specific platform
- Future OpenMP:
 - Target hints by platform/device
 - Increase flexibility of some constructs

Implementations Make Different Assumptions

- CPUs prefer:
 - Using simd where possible
 - Using static but un-chunked schedules
- GPUs prefer:
 - Avoiding simd construct
 - Using `schedule(static, 1)`

Gaps for performance portability with OpenMP 4.X

Major Follow-ups: Part 1

- Deep copy
- Support for mapping of polymorphic objects
- Data transformation (serialization/deserialization-like) interface
- Mechanisms to support out-of-core streaming and pipelining of same
- First-class template/lambda support

Major Follow-ups: Part 2

- Portable interfaces to access and manipulate:
 - Dependencies, including native mechanisms for same
 - Target device function pointers
 - Closure-like constructs for outlined functions
- Reduce the dependence on declare target:
Support inline/template/constexpr
- Top-level asynchronous tasking
- Event-loop programming model support

Major Follow-ups: Part 3

- Multi-level memory management (HBM/GDDR/etc.)
- Support for “flexible” parallelism, or perhaps less restrictive collapse?
- Hints for declare target to express parallelism in functions outside the translation unit

